LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-675479

Connecting Performance Analysis and Visualization to Advance Extreme Scale Computing

P. Bremer, B. Mohr, M. Schulz, V. Pascucci

July 29, 2015

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Connecting Performance Analysis and Visualization to Advance Extreme Scale Computing

Report Driven by the Dagstuhl Seminar 14022
January 6th-10th, 2014

Seminar Organizers:
Peer-Timo Bremer (LLNL, bremer5@llnl.gov),
Bernd Mohr (JSC, b.mohr@fz-juelich.de),
Valerio Pascucci (SCI/Utah, pascucci@sci.utah.edu),
Martin Schulz (LLNL, schulzm@llnl.gov),

Report Contributors:
Todd Gamblin (LLNL, tgamblin@llnl.gov),
Holger Brunst (Technische Universität Dresden, holger.brunst@tu-dresden.de)

1 Vision for Combining Performance Analysis and Visualization

The need for predictive scientific simulation has driven the creation of increasingly powerful supercomputers over the last two decades. No longer the simple, single-processor machines of the 1970's and 1980's, modern supercomputers comprise millions of cores connected through deep on-node memory hierarchies and complex network topologies. In contrast to the direction that mainstream application development has taken, where rapid development is prioritized and hardware details are often an afterthought, simulation science thrives only through high performance. Increasingly, *more* knowledge of esoteric hardware details is required to exploit the performance of modern machines. The algorithms implemented by large simulations are already dauntingly complex, and the set of skills required to integrate domain science, numerical algorithms, and computer science is easily beyond the capability of any single scientist.

Performance analysis is a subfield of computer science that, for many years, has focused on the development of tools and techniques to quantify the performance of large-scale simulations on parallel machines. There are now a number of widely used tools and APIs to collect a wide range of performance data at the largest scales. This includes counts of micro-architectural events such as cache misses or floating point and integer operations, as well as timings of specific regions of code. The success of these tools has created a new challenge: the resulting data is too large and too complex to be analyzed in a straightforward manner. Existing tools use only rudimentary visualization and analysis techniques. They rely on users to infer connections between measurements and observed behavior. The raw data is abstract and unintuitive, and it is often poorly understood as much of the hardware details are undocumented by vendors. Automatic analysis approaches must be developed to allow application developers to intuitively understand the multiple, interdependent effects their algorithmic choices have on the final performance.

The natural first step towards automatic analysis is to visualize collected data. This provides insight into general trends. Visualization helps both application developers and performance experts form new hypotheses on causes of and solutions of performance problems. The HPC community has traditionally been associated with researchers in *scientific* visualization, but performance data is not necessarily a good fit for this model. The data is non-spatial, highly abstract, and often categorical. While some early attempts at including visualizations in performance tools have been proposed, these are rudimentary at best and have not found widespread adoption. The information visualization (info-vis) community is growing rapidly, focused on developing techniques to visualize and analyze complex, non-spatial data. There is a large body of work on general visualization design principles, color spaces, and user interfaces as well as a wide array of common techniques to tackle a broad range of applications. Unfortunately, there has so far been little overlap between the info-vis and performance analysis communities.

This Dagstuhl Perspectives Workshop, for the first time, gathered leading experts from both the fields of visualization and performance analysis for joint discussions on existing solutions, open problems, and the potential opportunities for future collaborations. The week started with a number of keynote sessions from authorities from each field to introduce the necessary background and to form a common baseline for later discussions. It became apparent that there is a significant overlap in the common tasks and challenges in performance analysis and the abstract problem definitions and concepts common in visualization research. The workshop continued with short talks focusing on various more specific aspects existing challenges and potential solutions, interspersed with increasingly longer group discussions. Theses extensive, inclusive, and in-depth exchanges ultimately shaped the second half of the workshop. This was only possible through Dagstuhl's unique collaborative discussion environment.

Ultimately, the workshop has spawned a number of collaborations and research projects between previously disparate fields, with potential for significant impact in both areas. Participants developed three high-level recommendations: First, joint funding for the various open research questions; second, support to build and foster a new community at the intersection of visualization and performance analysis; and third, the need to integrate performance visualization into the workflow of parallel application developers from design to optimization and production.

In this report we summarize the discussions and results of the Dagstuhl workshop, organized by the four major results the seminar led to:

- The attendees summarized the state-of-the-art and its gaps for both performance analysis and information visualization. At the same time, this information functioned as education for the attendees of the other field and provided an introduction into the respective fields (Section 2).
- The group outlined the common challenges in bringing the two fields together (Section 3).

- One of the major obstacles identified during this activity was the need for a common understanding of the data being collected and visualized, which then lead to the definition of a general data model and a discussion on how existing tools can be mapped to it (Section 4).
- Finally, the seminar produced a set of next steps and recommendations that will merging of the two fields and have the potential to significantly further the performance visualization for large scale systems (Section 5).

Overall, these results and the report based on them will result in closer collaboration between the fields of performance analysis and visualization, creating a vibrant new field of performance visualization.

2 Background

The characterization, modeling, analysis, and tuning of software performance has been a central topic in High Performance Computing (HPC) since its early beginnings. The overall goal is to make HPC software run faster on particular hardware, either through better scheduling, on-node resource utilization, or more efficient distributed communication. The first step in optimizing is typically to collect some data about the program's behavior at runtime. Collecting and displaying this data to diagnose particular types of performance problems is the forte of current tools. The difficulty, and a long-standing open problem in performance analysis, is that for large parallel programs, there is simply too much performance data. Even a simple, single-threaded sequential program run on one processors can generate instruction traces comprising *millions* of execution events.

In a large parallel program, where data is collected from multiple threads on each node and potentially hundreds of thousands of nodes, the data becomes extremely unwieldy. Locating and identifying a performance culprit in such data is often like finding a needle in a hay stack, and it is typically not known in advance which data might help in the diagnosis, or which processes/threads it might be from. Visualization has therefore been used for the initial, high-level analysis of performance data for purely practical reasons: we delegate the task of finding the problem to the user.

During the seminar it became apparent that HPC performance analysts and InfoVis experts are both dealing with *visualizing and analyzing complex data*, and that there is great potential for collaboration between the two fields. InfoVis experts have developed many techniques to extract meaning from very high dimensional unstructured data, and to search it for correlations. Performance analysts have a strong focus on very large, relatively structured data sets that need to be studied in light of irregularities or known deficiencies. Much of this data, particularly with respect to networking and performance counters, is high dimensional and unstructured.

This section summarizes the state-of-the-art analysis and visualization in the performance analysis field and provides background for the common data model we propose later in Section 4.

2.1 Profiling Tools

The most basic performance measurement tools are *profilers*, which record, for some execution of a program, the parts of the source or binary code in which a program spent its time. A profile is essentially a histogram, binned by region in the code, of elapsed time spent executing. The key difference among profilers is *how* they bin locations in the source code.

1. **Flat profilers.** So-called *flat* profilers consider source code entirely statically, and bin source code by function names, source file and line number, or raw instruction addresses in a binary. While this is a simple way to consider source code, it ignores the *calling context* of the code. For example, a flat profiler will accumulate time into a single bin for the `MPI_Send()` function regardless of whether it was called directly by the application or from a library. Simple call graph profilers like IBM's Xprofiler include static call information in their display, but they are still flat profilers because they do not include any dynamic calling context information.
2. **Call-tree profilers** add information to a flat profile by binning code by calling context, i.e., by the contents of the runtime stack it was called from. The unique bins in such a profile comprise a *calling context tree* or *call tree*, typically rooted at the `main` function or one of its ancestors. Binning based on dynamic runtime information can differentiate different uses of the same call, especially for library functions. It can also differentiate uses

of different functions through the same callsite, e.g., in the case where a function is called indirectly through a pointer. Open|SpeedShop, Scalasca, TAU, and HPCToolkit all support this mode of profiling.

3. **Function profilers** such as mpiP intercept and time only certain function calls. mpiP in particular is a profiler tailored to the MPI interface, and because it only accesses certain functions, it can bin MPI operations according to their parameter values in addition to calling context. It is, however, unaware of time spent out side the MPI interface. Darshan is another example of an interface profiler that targets POSIX I/O commands.

In addition to the way they account for source locations, profilers differ in the metrics they can display, i.e. the metrics accumulated in each bin of the histogram. For example, HPCToolkit and Open|SpeedShopcan both be configured to accumulate floating point instruction executions, cache misses, or other events instead of simply CPU cycles or time. In practice, profilers incorporate many aspects of the above traits. For example, mpiP incorporates calling context in addition to function parameters, and a call tree profiler can be used to generate a flat profiler with post-processing.

Handling parallelism When profiling a parallel program each process or thread typically generates its own copy of the profiling histogram. The task of aggregating profile data in large parallel programs is currently either handled by writing a single file per monitored task, or by aggregating this data. Parallel statistical techniques such as clustering have been proposed, but none of these techniques is used consistently in production tools to isolate problems, at least not for simple profiling. More often, data is averaged or summed across all processes, losing load imbalance information and any other heterogeneity in the profiled data.

Visualizing profiles The profilers above display their results in many different ways. HPCToolkit, Open|SpeedShop, Scalasca and TAU offer GUI tree widget views that show source locations and the percentage of total time spent in each. These views allow users to collapse and expand nodes of the calling context tree. Flat profilers typically provide a simple table, possibly grouped into categories for organization, e.g., by library or by source file. mpiP and Open|SpeedShopprovide simple human-readable text output, and Darshan provides a web interface for visualizing its results. Some profilers provide mechanisms for zooming in on data based on some relevance criteria, for example, HPCToolkit allows a user to zoom in on the ”hot path” in the profile by iteratively expanding a child with a large percentage of total time up to some threshold. IBM's XProfiler and TAU's callgraph view adjusts the size of call graph nodes based on the time spent in them, but aside from this, none of the current crop of tools provides visual metaphors to emphasize particular nodes in the tree or graph beyond simple coloring.

2.2 Tracing Tools: Performance Timelines

Tracing tools record a *sequence* of parallel events per process. These events might include entering a particular function, exiting a function, sending a message, receiving a message, etc. Each event is recorded, typically with a timestamp, and possibly also with a set of metric counts for the time of the event. Unlike a profile, where individual event records are discarded to conserve space and to build up an aggregate histogram, traces store the *entire* event trace. They can thus grow much larger than profiles, but can reveal phenomena that profiles cannot.

Traces are useful when program behavior depends on temporal information. In these cases, determining the root cause of a problem from a profile may not be possible. Conditions depend on the order of events as they execute at runtime, and the performance of many distributed communication operations depends on the order in which messages are passed among parallel processes. In thread-parallel programs, shared memory data exchange and locking is used for communication and synchronization, but the idea is the same.

Tracing tools are most typically used with MPI parallel applications. JumpShot, VampirTrace, Scalasca, Score-P, and TAU all provide the ability to measure the time an application enters and exits MPI calls, as well as the endpoints of parallel communication operations like sends and receives. Many also provide instrumenting compilers that allow the entries into and exits from local computation routines.

Handling parallelism. Tracing tools, like profilers, record a series of events on every parallel task. Depending on the granularity of measurement, traces can grow very large very quickly, as their space complexity is proportional to the product of execution time and event frequency. Aggregating large amounts of trace data is difficult, though some

tools like ScalaTrace have been developed to analyze trace data for similarity for purposes of compression (but not for analysis). Most trace tools simply dump per-process, compressed trace records, which makes their scalability lower than that of profilers for common usage.

Visualizing Trace Data Trace data is typically visualized as a long timeline, or Gantt chart, with time on the horizontal axis and tasks (processes or threads) on the vertical axis. Colors are used to denote different communication and computation routines, and messages are shown as lines drawn between processes. Users can zoom into sub-regions of the full trace to expand complex behavior on short time scales. JumpShot and VNG both provide this type of visualization for trace data. ScalaTrace does not provide a visualizer. The HPCTraceViewer tool combines call tree profiling with tracing by displaying a fully dynamic call tree event view. This is a two-dimensional view of the 3D space of tasks vs. time vs. call trees, and the user can select a call tree level in advance and view the traditional 2-D trace view. Each element in this trace view is a sampled call path – there are no metrics recorded on the callpath elements in HPCTraceViewer's view.

2.3 Traditional Plots

In addition to profiles and traces, standard two- or multi-variate plot-based visualizations have been employed by performance tools. Bar charts, pie charts, and standard histograms are used by TAU, Vampir, ParaProf, and Cube to visualize binned data in arbitrary metric domains. Scalasca can use BoxPlots to show variation and distribution of timing and other metric values.

Often, a plot is the best way to display a relationship between a small number of values, but the user must know in advance what to plot, and exploring all possibilities is typically tedious. Visualization and analysis techniques are typically necessary to guide the user towards the right set of metrics to plot. PerfExplorer allows curves to be fit to scatter plots, and it can automate the generation of large numbers of plots for high-dimensional data, but the user must still scroll through a large array of bad curve fits to find the interesting ones.

2.4 Projected Views

Recently, many performance tools have begun to explore the idea of mapping, or *projecting* performance data onto spatial, logical, or other domains to show correlations and topological relationships.

The Cube tool allows a user to visualize metrics associated with each process in a large, parallel Blue Gene, Cray or K computer job to be displayed in the logical topology the tasks comprise at runtime. Blue Gene machines, as well as other supercomputers, employ cartesian *torus* or *mesh* shaped networks, and this view allows us to visualize processes in a projection that clearly displays their communication locality.

The PAVE project at LLNL has projected performance data into the simulated application domain, then used traditional scientific visualization techniques to display the resulting data. This revealed that for some fluid dynamics codes, there are correlations between performance metrics and particular domain data. Scalasca has been used to show a similar visualization of climate simulation data projected onto a visualization of the globe, and the TAU tool provides a similar projection tool that allows data to be projected onto an arbitrary 3D geometric shape to show correlations. These techniques allow users to understand data-dependent performance problems, and to identify what part of the source code cause data-dependent delays in the overall computation.

The Boxfish tool at LLNL was developed to generalize the idea of projecting performance data onto domain-specific views. For example, given a rendering of a 2-D, 3-D, or 5-D torus network, Boxfish can display per-process data projected onto the nodes and links of the network. It can also project the same data onto a custom 2-D network visualization, and display the same data with less clutter. Or, it can project this data onto a representation of the simulated physical domain, such as a material patch view for AMR applications. Boxfish is structured so that the elements of the view to be colored or labeled are exposed for *any* performance data to be projected, which allows a user to explore correlations and relationships between performance measured in one domain with elements in some other domain, assuming a suitable view plugin has been developed for the domain.

2.5 Information Visualization for Performance Analysis

Most of the visualization techniques discussed above have been driven directly by specific needs of performance analysis. This has made these tools intuitive to use and well integrated into the HPC workflow. Unfortunately, these visualizations rarely consider aspects such as appropriate color selection, screen space usage, more abstract visual metaphors, or the advantage of interactive linked-view interfaces. This often results in cluttered displays, misleading visualizations, unscalable representations, and static plots not suited for an interactive exploration driven by the end user.

On the contrary, the field of information visualization has a long history of research in all these aspects. For example, there exist a number of taxonomies to classify different visualization tasks [4, 12, 9], extensive research on the use of color in visualization [2, 3, 10], and accepted practices on the design of visualization systems [5]. Unfortunately, due to the large amount of domain knowledge necessary to collect and interpret performance data only comparatively little research in the information visualization community has focused on performance data. Nevertheless, these include a number of interesting new concepts such as novel layouts for networks [11, 8] or the memory topology [6] and a different perspective on how to depict time [15]. However, these tools have found only very limited adoption as they either do not directly or completely address the users needs, are limited in scale, or have simply proven to unintuitive for most HPC researchers. As a result a recent survey in related work on performance visualization [7] finds a large body of research that appears split into a set of widely used tools with often limited visualization capabilities and a set of advanced techniques, graphical layouts, and systems that is predominantly academic in nature.

2.6 Future Direction for Performance Visualization

One immediate result of even the first day of the Dagstuhl workshop was the realization that the combination of advanced visualization techniques with state of the art performance analysis technology has the potential for significant impact in virtually all areas of HPC. However, it also became clear that each area individually will have difficulty addressing the open challenges adequately. Instead, we need an exchange of knowledge in both direction with the visualization community becoming more familiar with HPC problems, techniques, and existing tools and the performance analysis community adapting more advanced visual encodings, integrated interfaces, and interactive tools. The data model described in Section 4 is a first step in this direction by describing, for the first time, a holistic view of performance data starting from the collection of raw measurements and ending and interactive visualization tools. As discussed in Section 4.6 many of the existing tools are well described by this model and recent developments designed to project data from one domain into the other can be see as an initial attempt at a complete realization of this model. Nevertheless, during the workshop it became apparent that to allow the techniques used in scientific and information visualization to be applied to performance tools, there will need to be more common nomenclature and more standardization in the way performance data is stored and exchanged.

3 Challenges

One of the central challenges of parallel performance analysis is the extreme volume and variety of measurements that can be gathered from parallel performance tools. Performance analysts have struggled with devising ways to gather and analyze this data for many years. Information and Scientific Visualization techniques have the potential to offer performance analysts an entirely new set of tools to analyze this data, but it was determined at the seminar that several obstacles impede collaboration between the performance analysis and visualization communities.

Domain Knowledge First, there are knowledge and terminology gaps. Performance tools are often designed for experts. Understanding the measurements they generate can require a full-stack understanding of a supercomputer, making them opaque to most visualization experts. For example, identifying code regions may require knowledge of compilers and language implementation, as this determines how symbols are represented at runtime. Further, understanding many measurements, e.g., network performance measurements, likely requires knowledge of the network on a particular supercomputer machine. Finally, understanding the types of analyses likely to reveal performance problems typically requires knowledge of how particular instructions and code executes on specific hardware.

Measurement Complexity In addition to complex data formats, complexities of performance measurement have made it difficult to easily relate data from one tool to that of another. As mentioned, tools like profilers typically aggregate data in very specific ways, e.g., by building a histogram according to a code representation with a specific granularity. Other tools, such as AutomaDeD, may aggregate data on the fly at runtime, e.g., to build a stochastic model of control flow or to only record timing for specific functions (e.g., MPI calls). There are no readily available tools to relate aggregate data to exhaustive data, or to project (lossy or otherwise) data in one domain to another.

Data Formats Because of the complexity of performance measurement, data collected by existing performance measurement tools are often stored in a manner closely tied to a particular performance tool's implementation. This is done for efficiency, either because the format was the easiest structure to use for *collecting* measurements, or simply because the designers of the measurement tool only envisioned their data being consumed by a single, bespoke performance visualization tool. The profusion of performance tool formats and the lack of documentation on them for non-experts makes exchanging data with visualization researchers hard, as accessing the data may require implementing a new parser or translator to even begin working with the data.

Evaluation Once the collaboration between performance tool developers and information visualization experts is successful, and new data analysis or visualization functionality is added a performance tool (prototype), the challenge remains how to evaluate the effectiveness of the new approach. Given the extreme diversity of parallel machine and application architecture and the needs of application domains, it will be very difficult to assess the usefulness of a new approach. If a new approach was successful highlighting a performance problem in one or two cases does not automatically mean it is useful in general. On the other hand, if various experiments with a new approach have not produced any interesting performance result, does not mean the approach is not useful: the experiments might just have picked the wrong application examples or parameters which (by luck) just work fine. Developing general purpose metrics and benchmarks for evaluating new approaches for parallel performance methods and tools will be difficult.

4 The Foundation: A Common Data Model

One of the most significant results of this Dagstuhl seminar has been the joint realization that virtually all existing performance data collection and analysis routines can be defined in and described by a rather simple, yet general data model. Both performance analysts and visualization experts agreed that formally specifying a data model with which to describe performance results would allow for better communication and data exchange between the two communities. Making data more accessible, both in terms of a common format and in terms of the common structure needed to *explain* the format to others, will lower significant barriers to collaboration. In addition to the obvious benefit of giving InfoVis researchers access to performance data, it will facilitate exchange of data among performance researchers. This has the potential to allow measurements from different tools to be combined in ways not possible before, enabling new types of performance analysis.

We expect that a common data model will also go a long way towards solving the first problem of domain knowledge. With a common language to describe performance data it will become easier to understand for visualization experts. The resulting collaborations will serve to educate both communities about how best to visualize, analyze, and understand this data. The key, as this seminar has demonstrated, is lowering the barriers to entry that obstruct the speedy exchange of information.

4.1 The Anatomy of Performance Data

At a high level, parallel performance data can be said to describe the state of a dynamic system, usually a parallel supercomputer or some part of it. Typically, this data is recorded at runtime during the execution of some parallel application, as this is how most performance tools store their data. However, performance data may also include data recorded by tools or system logging daemons that run continuously, and that persist beyond the lifetime of a single parallel run. Further, it may include data that describes properties of the system or application — that is, metadata.

Structurally, there is no particular distinction between recorded data and metadata. Semantically, metadata typically describes some static aspect of the system, like the dimensions of a network, or system configuration information particular to a single application execution, and recorded data represents dynamic measurements.

4.2 Notional Example

Before we delve into the details of the model, we will start with a notional example to illustrate the kinds of data we are dealing with. Suppose we have a performance tool that records, for each function in some application, the time spent in the function during each time step, the number of floating point operations executed by the function, and the number of cache misses incurred by the function's execution. We might represent this as a simple table:

FunctionName	Time	FP	CM
hydro	395	452	64
checkpoint	12342	0	7249
solve	19234	2097	

Each row in this table is a **record** written by some process. If we wanted to also record the 3-D torus network coordinates of the node where the measurement was taken, along with its rank in the MPI process, we would add columns to the table to get data that looks like this:

FunctionName	x	y	z	rank	Time	FP	CM
hydro	0	1	2	0	395	452	64
checkpoint	0	1	2	0	12342	0	7249
solve	0	1	2	0	19234	2097	

We can now see above that each record was taken on rank 0, which is located on node $(0, 1, 2)$ on the network. If we wanted to measure this data over multiple time steps we would further disambiguate them with a time step column, e.g.:

FunctionName	Time	x	y	z	rank	Time	FP	CM
hydro	0	0	1	2	0	395	452	64
checkpoint	0	0	1	2	0	12342	0	7249
solve	0	0	1	2	0	19234	2097	
hydro	1	0	1	2	0	395	452	64
checkpoint	1	0	1	2	0	12342		7248
solve	1	0	1	2	0	19224	2107	224

This is a very general representation of the data. Each record represents a single fact recorded about the system. We might read the first record as the fact that "the hydro function took 395 nanoseconds, executed 452 FLOPS, and incurred 64 cache misses, on rank 0, on node $(0, 1, 2)$". Each record is simply a tuple of related *attribute values*. An **attribute** is simply the name by which we refer to a column header, along with an associated **type** for it values. Here all of the attribute have integer-types values, except FunctionName, which uses a string. A type can be a primitive type, such as an integer or real number, or it can be a structured data type as might be commonly used in programming languages. We do not restrict the types of attribute values, but in practice they are likely limited by the data store used. For each measurement, not every attribute must have a value. Tools are free to store as little or as much data as is available when they take their measurements.

4.3 A Generalized Data Model for Performance Tools

Following this notional example, we develop a generalization and formalization of our data model. Figure 1 provides a high-level sketch of the concepts explained below.

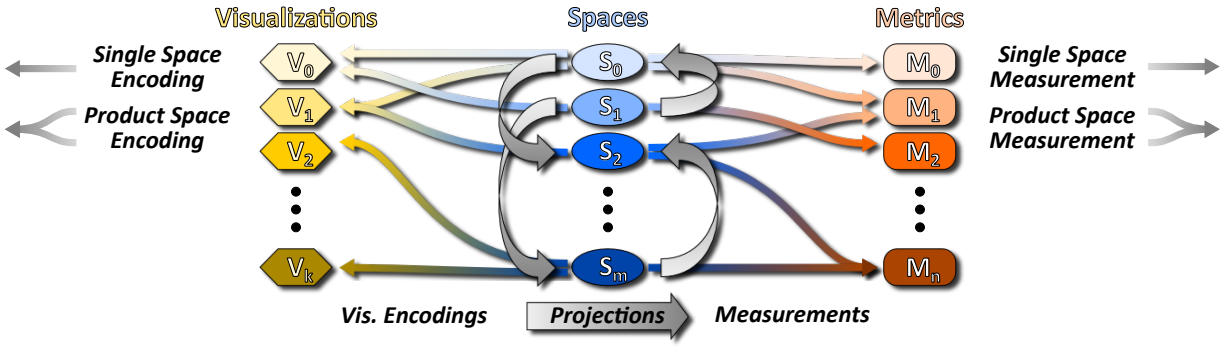


Figure 1: A generic data model that captures the relationships between metrics being collected by a performance tool, the spaces on which theses metrics are defined and the visualization and analysis tools to explore the data. Performance tools typically implement a *measurement* that collects metrics such as FLOP counts on some domain defined by either a single or a cross product of *spaces*. Visualization tools are tailored to (cross-products of) spaces, i.e., the MPI rank space for communication graphs, and can analyze data defined on the corresponding domain. To expand the type of data applicable to any one analysis technique the model contains *mappings* between spaces.

4.3.1 Spaces

At the core of the abstraction are a set of spaces. Each space is represented by a finite set of tuples and has a crossproduct of types associated with it, such that each type describes one element of the tuple. For example, the MPI rank space used above, uses a single integer to describe a location with respect to the communication graph and the triple of *x,y, z* coordinates describes the physical coordinates in a torus network. The number of spaces is not limited. Time and code (represented by calling context trees [16]) are treated the same as any other space.

4.3.2 Metrics

Metrics are units for individual data points. Examples are floating point operations and MPI message counts. Metrics are typically represented by infinite sets, as not to restrict what can be measured, but may in individual cases be a finite set of possible outcomes.

4.3.3 Measurements

Measurements capture the data acquisition in performance tools. They are represented as mappings of a crossproduct of spaces – the domain the performance data is collected in – to a metric, the set of possible values for this measurement. To make reasoning about measurements easier, we define a measurement as a unique mapping or function, i.e., for each element of the measurement domain the measurement only maps to at most one element in the metric set. If this is not the case for an experiment, e.g., in tracing tools that provide multiple data points for each element of a space over time, the domain needs to be modified to allow for this uniqueness, in the example by adding a space representing real or virtual time to the crossproduct that forms the domain.

4.3.4 Comparison to Traditional Database models

Thus far our data model is extremely general, and is closely related to many data models already in the literature. In particular, our model is very similar to the time-honored relational model, used in many database management systems. In the relational model, data is grouped into tuples, much like our records. A **relation** is a set of tuples with particular, common attributes – a **table** in database terms.

One key difference between our model and the relational model is that we do not restrict all records to have the same set of columns. Our model is designed to consume data from many different performance tools, each of which

may have its own set of attributes, and each of which may even provide records with missing data. In this regard, we do not have *relations* in the same sense that a relational database does.

Recent NoSQL data stores such as Google's BigTable and Apache Cassandra are more flexible than a relational database with respect to columns. These stores represent their data as a large, multi-dimensional, sorted, distributed map. Our data most resembles this model because of the freedom to create new columns/attributes on demand and because of the lack of a rigid schema.

As will be described in more detail in the subsequent sections, a key aspect of our model is *projections* between data spaces, as well as the concept of *measurement functions*. While a traditional relational database is designed to do efficient queries on data with a known schema, our model may create new data by projecting existing values into new spaces, effectively creating new columns in the model. Adding attributes based on analysis is key to our model of visualizing performance data, and the fact that our model includes both structured computation in addition to data storage and retrieval differentiates it from the relational model, as well as recent NoSQL data stores.

4.4 Projections

In performance data analysis, and particularly in performance visualization, it is very useful to be able to project data from one domain to another. As mentioned in Section 2.4, several recent tools have begun to adopt projections and similar operations to visualize correlations between different types of performance data.

4.4.1 Projections in the Notional Example

Let us revisit our notional example and consider the following data, this time in terms of links rather than nodes on the network:

FunctionName	sx	sy	sz	dx	dy	dz	bytes
hydro	0	1	2	0	2	2	100
checkpoint	0	1	2	0	2	2	100
solve	0	1	2	0	0	2	50

Each of these records now represents bytes sent, e.g., to a neighbor in the torus network by a particular node in a particular function. A network link is identified by a source (sx, sy, sz) triple and a destination (dx, dy, dz) triple.

4.4.2 Formalization of Projections

A projection, in the sense of the data model, maps one or more spaces (the origin domain) to one or more spaces (the target domain). This allows measurements represented in the target domain to be used in analysis operations on the origin domain. In general, we can distinguish three types of projections, which are also illustrated in Figure 2:

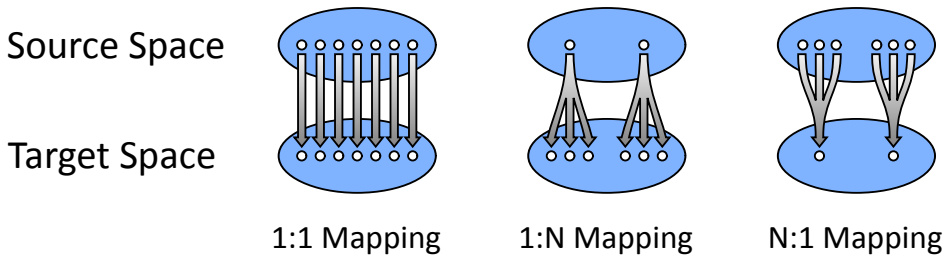


Figure 2: Three different types of projections between spaces.

- **1:1 Projections:** Each element of the origin domain is mapped to exactly one element of the target domain. An example of such a 1:1 projection is the mapping between node coordinates in a network to node IDs, since both domains describe the same physical entity, but using different names or numbering schemes. 1:1 projections allow a direct translation of measurements in one domain to another.
- **1:N Projections:** Each element of the origin domain is mapped to one or more elements of the target domain. An example for a such a 1:N projection is the mapping from node IDs in a system to process or MPI rank, since multiple ranks can be on each node. When mapping measurements using a 1:N projection, a measurement from an element in the target domain must be distributed or spread over all elements in the origin domain that map to it. The semantics of this operation depends on the semantics of the domains. For example, the same measured value could be attributed to each element in the origin domain in full, or the value could be split up based on a distribution function.
- **N:1 Projections:** Each element of the origin domain is mapped to at most one, not necessarily unique, element of the target domain. An example for a such a N:1 projection is the mapping of MPI ranks to nodes in a system, since multiple ranks can be on each node. When mapping measurements using a N:1 projection, measurements from all elements in target domain that map to a single element in the origin domain have to be combined using an aggregation operation. This can be as simple as a sum or average, but can also be a more complex operation such as clustering or statistical outlier detection.

Projections can further be combined into new projections, allowing a translation over multiple domains from an origin to a target domain. This could also lead to situations in which multiple translations between two domains using different compositions, i.e., a different route through the set of available domains, are possible. Note, though, that not all combined mappings between the same domains carry the same semantics. For example, a 1:1 projection between two domains may also be representable by a combination of a N:1 and a 1:N mapping, but the latter would include a loss of information by first aggregating measured values before spreading them out again. Choosing the right combination of projections based on the intended analysis is therefore crucial.

4.5 Contexts and Visual Representations

The previous sections describe three fundamental aspects that can be used to describe performance data: the space in which samples are taken which allows to identify and attribute samples to different software or hardware entities; the metrics which are collected to describe the behavior of the system; and the potential projections that allow comparing and correlating information related to different portions of the system. While having a generic and thus portable description of performance data solves only part of the overarching problem. In particular, given a specific problem or analysis task one still must decide what portion of the data to consider and in what *context*. We consider a context to be a (collection of) space(s) together with a means to analyze or visualize values defined on these spaces. A context may be as simple as a plot of, for example, FLOP count per MPI process or as complex as a network visualization.

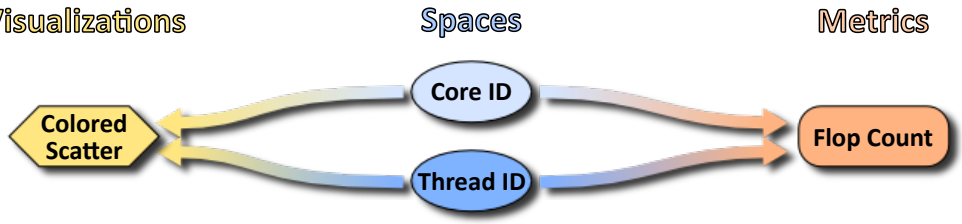


Figure 3: A simple context of a two-dimensional colored scatterplot with two spaces, core-id and thread-id, and a single metric FLOP count.

In particular, the more complex contexts such as a visualization showing the physical network have a *native* domain – in this case the node ids — and may require additional meta-information such as whether the network forms a mesh, a torus, a dragonfly etc.. Combined with the data model described above this now allows to describe a virtually arbitrary

analysis task as a simple combination of what metrics are of interest, on which spaces are the corresponding samples defined an in what context should these be analyzed. Figure 3 shows a diagram of a simple analysis such as a colored scatterplot expressed in this way. In the example, the FLOP count is recorded on the space of core-id and thread-id and the colored scatterplot context shows the data and potential relationships.

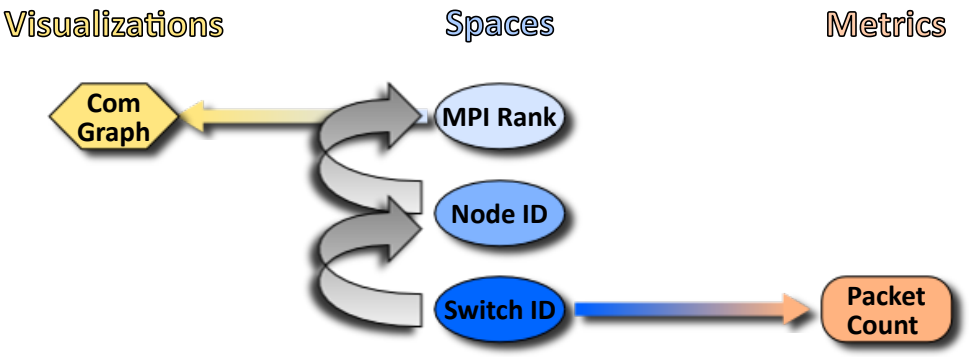


Figure 4: A more complex context example of a communication graph operating on MPI ranks that shows network packets collected on a per network switch bases. The count is first projected to node ids according to the hardware configuration and subsequently to MPI ranks via the node mapping file stored at runtime.

However, the true power of the new data model emerges when one considers projections. For example, Figure 4 shows a schematic of some context depending on MPI rank showing the number of packets per network switch. One way to analyze the packets in the context of the MPI ranks is to first project the samples from the space of the network switch to node ids and subsequently to MPI ranks. Assuming each node has its own switch the first projection is 1:1 but requires additional meta-information indicating which switch belongs to which node. The second projection from nodes to MPI requires the node mapping that was active for the corresponding run and is likely 1:N. Therefore, the user must decide how to process the packet counts further. For example, one could assign the packets evenly to all MPI ranks on a node or proportional to other say FLOP counts.

In the language of this data model, open challenges in performance analysis can now be classified as one of three areas: first, how to collect data, i.e., how to define measurements and collect samples in an efficient and scalable manner; second, how to attribute and connect data from different spaces, i.e., how to project samples collected on one space, e.g., AMR patch id, into another, e.g., core id; and third how to analyze or visualize data, i.e., by developing new useful contexts. Furthermore, the data model now decouples these tasks to the extent possible. For example, developing a new context like a new visual metaphor for a complex network topology becomes a well defined and independent task with a clearly defined native domain on which is operates. Any samples that can be projected into this native domain (potentially another independent research challenge) can use this context and thus tools and techniques can easily be combined. Finally, the data model is well suited to support the complex interconnected analysis likely to be required to understand future systems. Through the projections many tools, data sources, and contexts can be combined and connected, for example, to form the linked viewed interfaces an advanced visual analytics solution.

4.6 Examples of Existing Tools

The model introduced above can be used to reason about any kind of performance data. In the following we describe how it can be mapped to a selection of existing tools and how it can be used to describe their data. These tools cover the three major types of performance measurement approaches (sampling, profiling, and tracing) showing the generality of the base model.

4.6.1 Open|SpeedShop (Sampling)

Open|SpeedShop is a performance tool set, which includes both tracing and sampling experiments within a single tool and workflow [14]. For the following, we concentrate on the sampling experiments (the tracing parts are equivalent to

the Vampir tool covered below).

Sampling, or statistical sampling, is a technique to approximate the time spent by an application in various code segments. The execution of an application is repeatedly interrupted using a timer interrupt and during every interruption the tool records the program counter the application was executing at that time [1] and attributes the time since the last interrupt to that program counter location. If this is repeated often enough and with a fine enough granularity between interrupts, one can achieve a fairly accurate overview of which code pieces (identified by their program counter location) are executed for how long. As such, sampling tools provide an easy and low-overhead way to gain an overview of an application's performance.

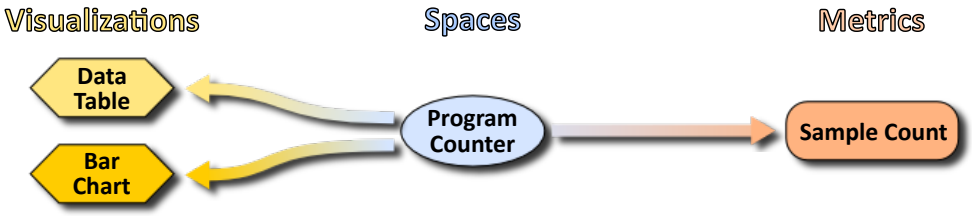


Figure 5: Open|SpeedShop described in the generic data model.

This kind of performance tool essentially provides a histogram of times attributed to each program counter location, or, in other words, an association of number of sample counts (represented by the time associated with them) to PC locations. This can be represented by a single space, covering the PC locations, and a metric space representing number of samples (see Figure 5). The data is then displayed as a big table or in the form of a bar chart. In both cases, the tool directly shows the histogram.

4.6.2 mpiP (Profiling)

mpiP is a profiling tool for MPI communication [17]. It tracks all calls to the MPI library, records the time spent in the MPI library and aggregates the information. At the end of the execution, mpiP then produces a report that shows the time spent inside the MPI library and in each type of MPI call or (when used together with stack trace information) each call path for all MPI call sites in the code. Additionally, mpiP keeps track of how much data is communicated and maps this data to the same entities as the timing data. Combined, this provides users with a basic, yet powerful overview of the communication characteristics of their application and can help to identify MPI routines that contribute to execution delays.

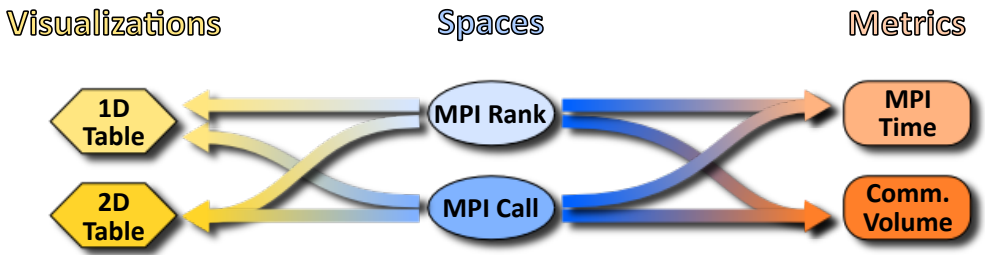


Figure 6: mpiP described in the generic data model.

Describing mpiP's data in the generic data model discussed above requires a more complex set of spaces. In its finest granularity data can be shown per MPI process (represented by the its rank in MPI_COMM_WORLD) and per MPI call. As shown in Figure 6, we represent both as a separate space and their crossproduct defines the domain for mpiP measurements. The latter has two metrics, one for time spent in MPI and one for communication volume, both

[1] More advanced sampling techniques also record the call stack and/or sample on other events than time, e.g., cache misses or remote memory access — with loss of generality, we limit our discussion here to pure program counter sampling to keep the description simpler.

defined on the entire domain. The visualization in the current tool is simple and consists only of text output of the raw data covering both dimensions of the underlying domain.

Additionally, mpiP reports the spent in each call site aggregated across all MPI processes. This can be represented in our general mode with a 1:N projection from the space representing call sites to the complete domain combined with an aggregation function, in this case a simple addition. This creates "implicit" measurements from the "call" domain to the same metrics. This information is again printed in textual form, but this time has just a single dimension. Going even further, mpiP also reports the total time spent (across all processes and across all call sites), which can again be represented with a 1:N mapping from a new domain, represented by a single identifier representing the application run, to the entire measurement domain. This allows us to extract the total time spent in MPI (and the total data volume) which is then simply printed as a scalar.

4.6.3 Score-P/Vampir (Tracing)

The tracer component of Score-P [1] is an MPI tracer that, like mpiP, intercepts and records every call to the MPI library and stores the collected data. Unlike mpiP, though, every invocation is stored separately without aggregation. This enables user to collect and subsequently analyze a complete trace of all communication events. Additionally, Score-P optionally uses compiler-based instrumentation to track invocations of functions and records this information in the same way as invocations to the MPI library. Both traces (computation functions and MPI invocations) are then combined and can be visualized using the Vampir tool set using a Gantt chart like display [13].

In order to represent the data in our model, we start with the model used by mpiP, but add additional spaces. Namely, Score-P also records a time stamp for each measurement as well as a full call path. These four spaces are combined into two different domains: The MPI Call $\times$ MPI Rank $\times$ Time domain describing which MPI rank communicated when and using what MPI call; and the Call Path $\times$ MPI Rank $\times$ Time domain, which describes in which function the communication occurred. Conceptually, all measurements taken are recorded with respect to both domains independently though for efficiency reasons they are likely stored more compactly. Both domains are then combined in a Gantt chart which typically uses the first domain as primary axes, showing MPI ranks on the y-axis and time on the x-axis and the second domain to color entries by various metrics or function types.

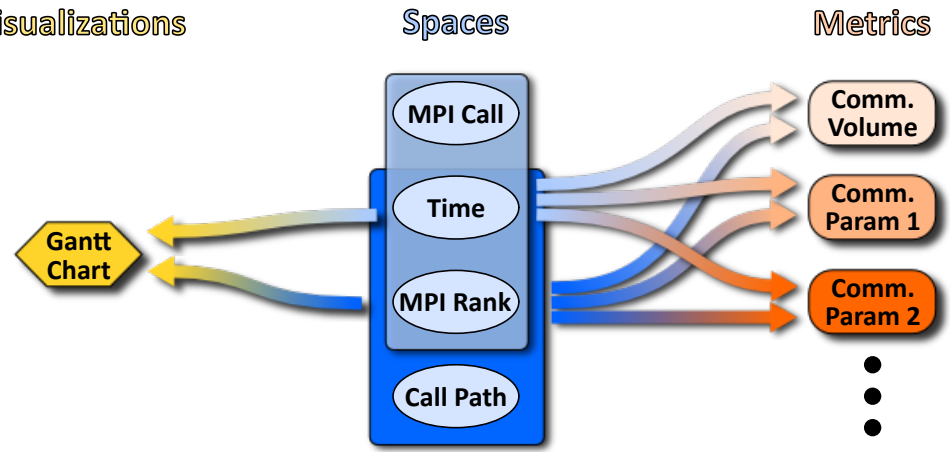


Figure 7: Score-P/Vampir described in the generic data model.

Note, that as with mpiP, Score-P/Vampir can aggregate data by any of the three dimensions, which again can be presented with corresponding 1:N projections. We omit these in the figure to keep the diagram legible, though. In particular, aggregating from the time space allows tracing data to be reduced to profiling data, equivalent to the data reported by mpiP. This is one example of how this model can help to map data between tools and their corresponding visualization approaches. Further, the "function" space used here and the PC space introduced earlier during the discussion of Open|SpeedShop, are related - each PC is associated with one function. We can therefore also provide

a projection between those two spaces, making it possible with our model to bride the gap between sampling and tracing/profiling tools.

4.7 Discussion

One of the immediate impacts already apparent during the workshop itself has been to provide a common framework for both communities to discuss ideas and solutions. In particular, the ability to describe existing tools on both sides in terms of the data model and the relevant projections has greatly facilitated communications. Even though the model will likely be refined by both communities in coming years this represents a significant and lasting impact. Furthermore, we believe that the model can serve as a blueprint for general, interoperable, and cross-community tools. The three main ingredients of the model, measuring data, visualizing/analyzing data, and projecting data directly correspond to a set of well defined and to a large extent independent tasks. Assuming appropriate interfaces, techniques and implementations in any of these three areas can be combined in a variety of ways. For example, many projections between non-trivial spaces, e.g., MPI rank vs. network port, require detailed machine-specific knowledge and sophisticated low-level tools to extract it. As a result this functionality will be challenging for visualization researchers to implement. At the same time mappings are crucial to expand the type of data that can be display beyond the native domain of a visualization and to provide linked views. For example, understanding how a feature in the communication graph is expressed in or caused by the underlying network hardware could be tremendously helpful. However, this functionality requires the projection from MPI ranks to ports. Going forward we anticipate targeted tools to be developed for individual tasks, described with respect to the model, which allows them to be used more readily to assemble more powerful solutions.

5 Recommendations

By all accounts, the workshop has been a great success and has already led to a number of new collaborations, joint proposals, and a very successful workshop at the 2014 Supercomputing conference with 18 strong submissions. The second installation of the workshop is scheduled for Supercomputing 2015 and we expect a similar response. Nevertheless, much remains to be done to establish a subfield of performance visualization at the intersection of HPC performance analysis and information visualization. In particular, we have two specific recommendations to advance this new area of research and a number of suggestions to ensure a broader impact of the resulting work

5.1 Recommendation I: Dedicated Funding for Joint Research

While all the participants of the workshop agreed that combining forces has significant potential for future impact, this only came after intense personal discussions to overcome “language” and “cultural” differences between the two communities. At this point the greatest concern is that the core competencies of both areas are too far apart to organically grow closer. In particular, finding funding for the required long term collaborative teams necessary to make progress will likely be difficult in the current climate. At this moment few solicitations in performance analysis will consider visualization research within scope nor will program managers or reviewers fully appreciate the resources required to design new visual metaphors or interactive tools or the potential impact of these efforts. Similarly, solicitations in the area of information visualization typically do not consider research in the specific application area to be within scope nor does the corresponding community have sufficient insight into HPC problems to recognize the need for new techniques to collect and organize performance data. Nevertheless, it is our opinion that ultimately progress will crucially depend on close collaborations between experts of both fields over long periods of time as the initial start-up cost of such an effort will likely be substantial. As a result we recommend the creation of a dedicated funding stream contributed from both communities specifically designed for collaboration in the area of performance visualization to foster and encourage the foundational research necessary to establish this subarea.

This should include not only the wide range of topics associated with the core topic of turning the massive amounts of performance data into insightful visual representations, but also include support areas such as more structured data acquisition, interoperable data stores, visualization toolkits specialized to performance data, or (semi-) automatic analysis algorithm to enable novel visualizations. Further, underlying to all these efforts should a goal of scalability,

in terms of number processing cores used by application, data volume collected and stored, as well as throughput of analysis steps.

As an additional remark, this workshop has shown that both communities are highly international covering multiple countries and even continents. This is already evident from the list of attendees and their widely varying geographic background. However, current funding opportunities are often highly localized and intercontinental funding is an absolute rarity, which hinders collaborations. Approaches to overcome this deficit would be extremely valuable in supporting this new field and would likely lead to a series of new and highly fruitful collaborations.

5.2 Recommendation II: Building a Community

Another important aspect of establishing performance visualization as viable subfield is the creation of a research community. Maybe the most significant outcome of the Dagstuhl seminar has been the personal connections between researchers of both communities. Unfortunately, only a small fraction of interested researchers could attend and organizing similar events in the future has the potential for significant impact by simply making researchers on either side aware of the needs and abilities of the other. Further, support for dedicated publication outlets, like the workshop at Supercomputing 2014 and 2015 by the Dagstuhl organizers, would make this area of research more attractive, especially to junior researchers.

Closely connected to the remarks on funding above, but going beyond the monetary aspect, this area more than many others would benefit substantially from international project support. HPC systems, especially at the high end, are often unique resources available only at a small number of location. Remote access by researchers in other countries is often a difficult and in some cases (e.g., Japan) even prohibited by law. Overcoming these restrictions and providing an avenue for international teams to exchange machine access, system knowledge and leverage software infrastructure could greatly improve the pace of research.

Finally, at this moment one of the greatest obstacle for new research is the difficulty in provide or getting access to meaningful data to drive the develop of new techniques. Therefore, establishing an infrastructure by which the new community could share data, problem descriptions, benchmarks, etc., would allow a much larger number of researchers to participate in the process.

5.3 Recommendations for Broader Impact

Improving the performance of large scale codes can have a significant positive impact on HPC centers and their users and performance visualization can contribute substantially towards this goal. At the same time, performance visualization (both of individual applications and complete systems and facilities) can help raise awareness for this problem in the first place. Performance analysis is often seen as a second class citizen, as an activity to be done after the development of the code is (near) complete. This limits the impact performance analysis can have and results in inefficient executions and wasted performance or throughput — we get less science done than we could/should for the huge investments made in HPC centers.

By providing user friendly and intuitive representations, application developers are more likely to overcome the initial startup barrier and learning curve conventional tools often have and facilities can get an easier overview on how well their systems are used by which codes and have a chance to react. Such raised awareness for the need of performance analysis and optimization coupled with a new generation of tools that helps address existing performance bottlenecks, will go a long way in improving the efficiency of our large scale compute resources and will ultimately be critical to the underlying computational missions.

References

[1] Score-p user manual. https://silc.zih.tu-dresden.de/scorep-current/pdf/scorep.pdf, 2015.

[2] D. Borland and R.M. Taylor. Rainbow color map (still) considered harmful. *Computer Graphics and Applications, IEEE*, 27(2):14–17, March 2007.

[3] David Borland and Alan Huber. Collaboration-specific color-map design. *IEEE Comput. Graph. Appl.*, 31(4):7–11, July 2011.

[4] Matthew Brehmer and Tamara Munzner. A multi-level typology of abstract visualization tasks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2376–2385, December 2013.

[5] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[6] A. N. M. Imroz Choudhury and Paul Rosen. Abstract visualization of runtime memory behavior. In *VISSOFT*, pages 1–8. IEEE, 2011.

[7] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer. State of the Art of Performance Visualization. *Comput. Graph. Forum*, pages 141–160, 2014.

[8] A.G. Landge, J.A. Levine, K.E. Isaacs, A. Bhatele, T. Gamblin, M. Schulz, S.H. Langer, P.-T. Bremer, B. Hamann, and V. Pascucci. Visualizing network traffic to understand the performance of massively parallel simulations. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2467–2476, 2012.

[9] Zhicheng Liu and John Stasko. Mental models, visual reasoning and interaction in information visualization: A top-down perspective. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):999–1008, November 2010.

[10] Kenneth Moreland. Diverging color maps for scientific visualization. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II*, ISVC '09, pages 92–103, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] Christopher Muelder, Carmen Sigovan, Kwan-Liu Ma, Jason Cope, Sam Lang, Kamil Iskra, Pete Beckman, and Robert Ross. Visual analysis of i/o system behavior for high-end computing. In *Proceedings of the Third International Workshop on Large-scale System and Application Performance*, LSAP '11, pages 19–26, New York, NY, USA, 2011. ACM.

[12] Tamara Munzner. A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):921–928, November 2009.

[13] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.

[14] Martin Schulz, Jim Galarowicz, Don Maghrak, William Hachfeld, David Montoya, and Scott Cranford. Open|speedshop: An open source infrastructure for parallel performance analysis. *Scientific Programming*, 16(2-3):105–121, 2008.

[15] Carmen Sigovan, Chris W. Muelder, and Kwan-Liu Ma. Visualizing large-scale parallel communication traces using a particle animation technique. In *Proceedings of the 15th Eurographics Conference on Visualization*, EuroVis '13, pages 141–150, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association.

[16] Nathan R. Tallent, Laksono Adhianto, and John M. Mellor-Crummey. Scalable identification of load imbalance in parallel executions using call path profiles. In *Proceedings of IEEE/ACM Supercomputing '10*, November 2010.

[17] Jeffrey Vetter and Chris Chambreau. mpiP: Lightweight, Scalable MPI Profiling. `http://mpip.sourceforge.net`.

Acknowledgements

Dagstuhl

Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-TR-xxxxxx).

Workshop Participants

The following section lists all participants of the Dagstuhl Seminar "Connecting Performance Analysis and Visualization to Advance Extreme Scale Computing", which was held January 6-10, 2014, all of which contributed to this report either by writing or through the extensive discussions during the seminar. We have grouped them by one of four self-assigned categories and for each include two keywords describing key projects or other relevant keywords.

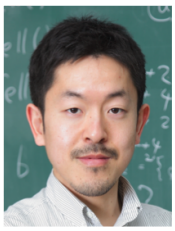
Developers of Performance Tools

Todd Gamblin
Lawrence Livermore National Lab

Keywords:
Boxfish
Scalable Tools

Markus Geimer
Jülich Supercomputing Center

Keywords:
Scalable Tools
Scalasca



Naoya Maruyama
RIKEN

Keywords:
GPUs
K Computer

Bernd Mohr
Jülich Supercomputing Center

Keywords:
Scalasca
TAU

Matthias Müller
RWTH Aachen

Keywords:
Virtual Reality
Validation Tools

Wolfgang Nagel
Technische Universität Dresden

Keywords:
Vampir

Martin Schulz
Lawrence Livermore National Lab

Keywords:
Scalable Tools
Boxfish

Felix Wolf
German Research School for Simulation Sciences

Keywords:
Scalasca
Cube

Developers of Visualization and Analysis Techniques

Remco Cheng
Tufts University

Keywords:
Interactive Visualization
Visual Analytics

Hans Hagen
Universität Kaiserslautern

Keywords:
Visualization
Applications

Daniel Keim
Universität Konstanz

Keywords:
Scalable Visualization
Visual Analytics

Joshua Levine
Clemson University

Keywords:
Topology
Boxfish

Klaus Mueller
Stony Brook

Keywords:
Visual Analytics
High Performance Computing

Valerio Pascucci
University of Utah

Keywords:
Large Data Visualization
Integrated Analysis

Carlos E. Scheidegger
AT&T Labs

Keywords:
Large Data Visualization
Graph Visualization

Tobias Schreck
University of Konstanz

Keywords:
Visual Analytics
Clustering

Derek Xiaoyu Wang
University of North Carolina, Charlotte

Keywords:
Unstructured Data
Mobile Data

Already Exploring the Overlap Between the Fields

Peer-Timo Bremer
Lawrence Livermore National Lab

Keywords:
Topology
Boxfish

Holger Brunst
Technische Universität Dresden

Keywords:
Vampir
Scalable Visualization

Hank Childs
Univeristy of Oregon

Keywords:
VisIt
TAU

Chris Muelder
University of California at Davis

Keywords:
Scalable Visualization
Visual Analytics

Judit Gimenez
Barcelona Supercomputing Center

Keywords:
Paraver

Application Developer/End User

Abhinav Bhatele
Lawrence Livermore National Laborator

Keywords:
Charm++
Projections

Hans-Joachim Bungartz
Technische Universität München

Keywords:
Scalable Algorithms
SPPEXA

Ulrich Rüde
Friedrich-Alexender Universität, Erlangen

Keywords:
Walbala
HHG